UNIVERSITEIT VAN AMSTERDAM

DEEP LEARNING

PRACTICAL 1

MLPS, CNNS AND BACKPROPAGATION

Andreea Teodora Patra
Student Number:13365169
andreea.patra@student.uva.nl

November 15, 2020

1 MLP backprop and Numpy Implementation

1.1 Evaluating the gradients

1.1.a Linear module

In this section, we are calculating the gradients for the linear module. We find closed form expressions for $\frac{\partial L}{\partial W}$, $\frac{\partial L}{\partial b}$, $\frac{\partial L}{\partial x}$.

$\mathbf{Y} = \mathbf{X}\mathbf{W}^T + \mathbf{B} \Rightarrow Y_{mn} = \sum_p X_{mp} W^T_{pn} + b_{mn} = \sum_p X_{mp} W_{np} + b_{mn}$

First we focus on the partial derivative with respect to the weights. Then, $\left[\frac{\partial L}{\partial W}\right]_{ij} = \sum_{mn} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial W_{ij}}$. We just need to calculate $\frac{\partial Y_{mn}}{\partial W_{ij}}$, because $\frac{\partial L}{\partial \mathbf{Y}}$ is provided by the next module during backprop.

$$\frac{\partial Y}{\partial W_{ij}} = \sum_p X_{mp} \frac{\partial W_{np}}{\partial W_{ij}} = \sum_p X_{mp} \delta_{ni} \delta_{pj} = \delta_{ni} X_{mj}$$
$$\frac{\partial L}{\partial W_{ij}} = \sum_{mn} \frac{\partial L}{\partial Y_{mn}} \delta_{ni} X_{mj} = \sum_m \left(\frac{\partial Y}{\partial Y}\right)^T_{im} X_{mj} \Rightarrow \frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{Y}}^T \mathbf{X}$$

Now, we calculate the gradient with respect to the bias term. Then, $\left[\frac{\partial L}{\partial b}\right]_{ij} = \sum_{mn} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial b_{ij}}$. We just need to calculate $\frac{\partial Y_{mn}}{\partial b_{ij}}$, because $\frac{\partial L}{\partial \mathbf{Y}}$ is provided by the next module during backprop.

$$\frac{\partial Y_{mn}}{\partial b_{ij}} = \mathbf{1}_{mn}$$
$$\frac{\partial L}{\partial \mathbf{b}} = \sum_{mn} \frac{\partial L}{\partial Y_{mn}} 1_{mn} \Rightarrow \frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{Y}} \mathbf{1}$$

Now, we calculate the gradient with respect to the inputs. Then, $\left[\frac{\partial L}{\partial X}\right]_{ij} = \sum_{mn} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial X_{ij}}$. We just need to calculate $\frac{\partial Y_{mn}}{\partial X_{ij}}$, because $\frac{\partial L}{\partial \mathbf{Y}}$ is provided by the next module during backprop.

$$\frac{\partial Y}{\partial X_{ij}} = \sum_p W_{np} \frac{\partial X_{mp}}{\partial X_{ij}} = \sum_p W_{np} \delta_{mi} \delta_{pj} = \delta_{mi} W_{nj}$$
$$\frac{\partial L}{\partial X_{ij}} = \sum_{mn} \frac{\partial L}{\partial Y_{mn}} \delta_{mi} W_{nj} = \sum_n \frac{\partial L}{\partial Y_{in}} W_{nj} \Rightarrow \frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \mathbf{W}$$

1.1.b Activation module

We consider an element wise activation module such that $Y_{ij} = h(X_{ij})$ and we need to calculate $\frac{\partial L}{\partial \mathbf{X}}$. Then, $\left[\frac{\partial L}{\partial X}\right]_{ij} = \sum_{mn} \frac{\partial L}{\partial Y_{mn}} \frac{\partial Y_{mn}}{\partial X_{ij}}$. We just need to calculate $\frac{\partial Y_{mn}}{\partial X_{ij}}$, because $\frac{\partial L}{\partial \mathbf{Y}}$ is provided by the next module during backprop.

$$\frac{\partial Y_{mn}}{\partial X_{ij}} = \frac{\partial h}{\partial X_{ij}} = \frac{\partial h(X_{mn}}{\partial X_{ij}} \frac{X_{mn}}{\partial X_{ij}} = \frac{\partial h}{\partial X_{ij}} \delta_{mi} \delta_{nj}$$
$$\frac{\partial L}{\partial X_{ij}} = \sum_{mn} \frac{\partial L}{\partial Y_{mn}} \frac{\partial h}{\partial X_{ij}} \delta_{mi} \delta_{nj} = \frac{\partial L}{\partial Y_{ij}} \frac{\partial h}{\partial X_{ij}} \Rightarrow \frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \circ \frac{\partial h}{\partial \mathbf{X}}$$

1.1.c Softmax and loss modules

Part i

We will consider the element wise activation function to be softmax and we will calculate its gradient with respect to the input. The softmax function is defined as follows $Y_{ij} = \frac{e^{X_{ij}}}{\sum_k e^{X_{ik}}}$. We need to calculate $\frac{\partial h}{\partial \mathbf{X}}$.

$$\begin{aligned}\frac{\partial h}{\partial X_{ij}} &= \frac{\partial h_{mn}}{\partial X_{ij}} \\ &= \frac{e^{X_{mn}}}{\sum_k e^{X_{mk}}}\left[\frac{\delta_{mi}\delta_{jn}\sum_k e^{X_{mk}} - e^{X_{mj}}\delta_{mi}}{\sum_k e^{X_{mk}}}\right] \\ &= \frac{\delta_{mi}e^{X_{mn}}}{\sum_k e^{X_{mk}}}\left[\delta_{jn} - \frac{e^{X_{mj}}}{\sum_k e^{X_{mk}}}\right]\end{aligned}$$

For this to work, we will need m=i all the time. Then, we evaluate 2 cases, one when j=n and one when $j \neq n$.

When j=n, we have:

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial Y_{ij}}\frac{e^{X_{in}}}{\sum_k e^{X_{ik}}}\left(1 - \frac{e^{X_{ij}}}{\sum_k e^{X_{ik}}}\right)$$

When $j \neq n$, we have:

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial Y_{ij}}\left(-\frac{e^{X_{in}}}{\sum_k e^{X_{ik}}}\frac{e^{X_{ij}}}{\sum_k e^{X_{ik}}}\right)$$

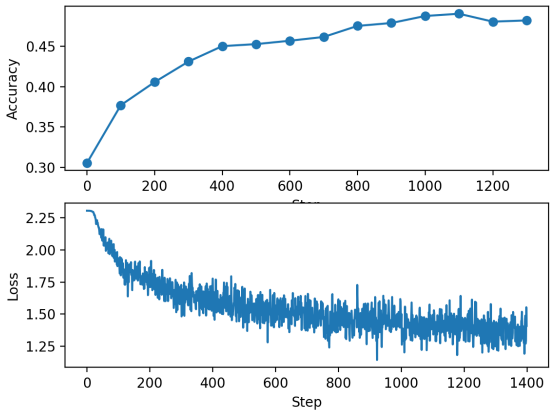
ii.

We will calculate the gradient of the crossentropy module with respect to the inputs. Then, we want $\frac{\partial L}{\partial \mathbf{X}}$ where $L = -\frac{1}{S}\sum_{ik} T_{ik} log(X_{ik})$.

$$\left[\frac{\partial L}{\partial X}\right]_{mn} = -\frac{1}{S}\sum_{ik} T_{ik}\frac{\partial log(X_{ik})}{\partial X_{mn}} = -\frac{1}{S}\sum_{ik} T_{ik}\frac{1}{X_{ik}}\delta_{im}\delta_{kn} = -\frac{1}{S}T_{mn}\frac{1}{X_{mn}} \Rightarrow \frac{\partial L}{\partial \mathbf{X}} = -\frac{1}{S}\frac{\mathbf{T}}{\mathbf{X}}$$

1.2 Numpy implementation

Using the gradient derived previously and an ELU activation function for all the Linear modules except the last one, we implement an MLP using only Numpy. The network is trained on CIFAR10 dataset and it has the following accuracy curves on the test set and the loss curves on the training set.

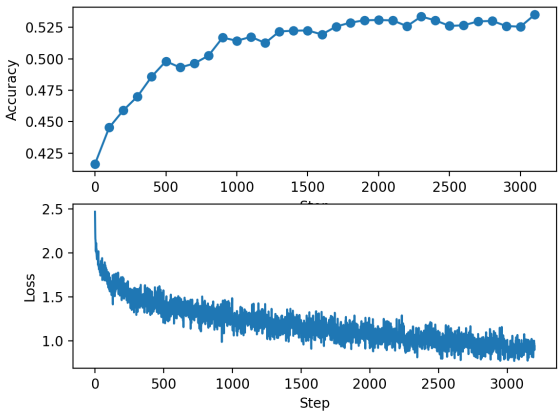


We can see the last accuracy reached is 0.48 on the test set using the default parameters. Overall, we can see that the loss converges between 1.25 and 1.5.

2 Pytoch MLP

2.1

Because we wanted to slightly increase the complexity of the network, we have changed the activation function to RELU and increased the neurons of the hidden layer to 200. When doing this, the accuracy has stayed the same at 0.4899. Then, we decided to train for more steps, and increased them to 3500. This time the accuracy increased to 0.5203. When running the network with an ELU activation function we get an accuracy of 0.5246. This agrees with the fact that ELU does not suffer from vanishing gradients compared to RELU. Finally, we also add BatchNormalisation after each liner module, which allows us to have a higher learning rate of 0.01. We also add another hidden layer with 200 neurons. When trying both RELU and ELU, it was found that RELU performs better. It has reached an accuracy of 0.5351.



We can see that our network has converged faster compared to the network with the default parameters and the final loss is in a smaller interval.

2.2

One of the benefits of tanh compared to ELU is the fact that it has centered nonlinearities and in this way it is easier to guarantee consistent behaviour. Moreover, tanh has a better output range compared to ELU which allows for stronger gradients and the gradient to tanh is a smooth function. Another disadvantage of ELU is that for positive big values, will always set the gradient to 1 compared to tanh which has more of a convergence to 0. One advantage of using ELU over tanh is that ELU doesn't suffer from vanishing gradients problem.

3 Costume Module: Layer Normalisation

Question 3.2a) We compute the gradients for the layer normalisation module.

$$\frac{\partial L}{\partial \gamma_i} = \sum_{sj} \frac{\partial L}{\partial Y_{sj}} \frac{\partial Y_{sj}}{\partial \gamma_i}$$

$$\frac{\partial Y_{sj}}{\partial \gamma_i} = \hat{X}_{sj} \delta_{ji}$$

$$\frac{\partial L}{\gamma_i} = \sum_{sj} \frac{\partial L}{\partial Y_{sj}} \hat{X}_{sj} \delta_{ji} = \sum_{s} \frac{\partial L}{\partial Y_{si}} \hat{X}_{si}$$

$$\frac{\partial L}{\partial \beta_i} = \sum_{sj} \frac{\partial L}{\partial Y_{sj}} \frac{\partial Y_{sj}}{\partial \beta_i}$$
$$\frac{\partial Y_{sj}}{\partial \beta_i} = \delta_{ji}$$
$$\frac{\partial L}{\partial \beta_i} = \sum_{sj} \frac{\partial L}{\partial Y_{sj}} \delta_{ji} = \sum_{s} \frac{\partial L}{\partial Y_{si}}$$

$$\frac{\partial L}{\partial X_{ri}} = \sum_{sj} \frac{\partial L}{\partial Y_{sj}} \frac{\partial Y_{sj}}{\partial X_{ri}}$$
$$\frac{\partial Y_{sj}}{\partial X_{ri}} = \gamma_j \frac{\partial \hat{X}_{sj}}{\partial X_{ri}}$$
$$\frac{\hat{X}_{sj}}{\partial X_{ri}} = \frac{\left(\frac{\partial X_{sj}}{\partial X_{ri}} - \frac{\partial \mu_s}{\partial X_{ri}}\right)\sqrt{\sigma_s^2 + \epsilon} - \frac{\partial \sqrt{\sigma_s^2+\epsilon}}{\partial X_{ri}}(X_{sj} - \mu_s)}{\sigma_s^2 + \epsilon}$$
$$\frac{\partial \mu_s}{\partial X_{ri}} = \frac{1}{M}\sum_{i=1}^{M} \delta_{sr} = \frac{1}{M}\delta_{sr}$$
$$\frac{\partial \sqrt{\sigma_s^2 + \epsilon}}{\partial X_{ri}} = \frac{1}{2\sqrt{\sigma_s^2 + \epsilon}} \frac{\partial \sigma_s^2}{\partial X_{ri}}$$
$$= \frac{1}{M\sqrt{\sigma_s^2 + \epsilon}} \Big[\sum_{p=1}^{M} (X_{sp} - \mu_s)\delta_{sr}\delta_{pi} - \frac{1}{M}\sum_{p=1}^{M}(X_{sp} - \mu_s)\delta_{sr} \Big]$$
$$= \frac{1}{M\sqrt{\sigma_s^2 + \epsilon}} \Big[(X_{si} - \mu_s)\delta_{sr} - \frac{1}{M}\sum_{p=1}^{M}(X_{sp} - \mu_s)\delta_{sr} \Big]$$
$$\frac{\frac{\partial \sqrt{\sigma_s^2+\epsilon}}{\partial X_{ri}(X_{sj}-\mu_s)}}{\sigma_s^2 + \epsilon} = \frac{1}{M\sqrt{\sigma_s^2 + \epsilon}} \hat{X}_{sj}\hat{X}_{si}\delta_{sr} - \frac{1}{M^2\sqrt{\sigma_s^2 + \epsilon}} \hat{X}_{sj} \sum_{p=1}^{M} \hat{X}_{sp}\delta_{sr}$$

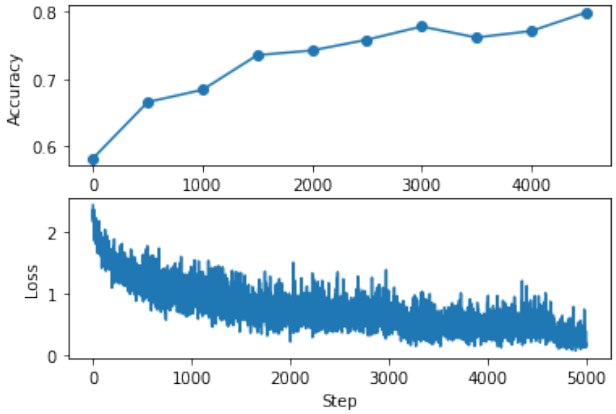
$$\frac{\partial L}{\partial X_{ri}} = \frac{1}{M\sqrt{\sigma_s^2 + \epsilon}} \Big[M\frac{\partial L}{\partial \hat{X}_{ri}}\gamma_i - \sum_j \frac{\partial L}{\partial \hat{X}_{rj}}\gamma_j - \sum_j \frac{\partial L}{\partial \hat{X}_{rj}} \hat{X}_{rj}\gamma_j \Big(\hat{X}_{ri} - \frac{1}{M}\sum_{p=1}^{M} \hat{X}_{rp} \Big) \Big]$$

Question 3.2d) First of all batch normalisation allows for higher learning rates, therefore faster convergence. We aim to have constant through time distribution of the input that is fed into each layer and using Batch Normalisation we can guarantee that after the backpropagation is performed the input distribution is preserved. Here, the mean and variance is calculated over the mini-batch whereas in layer normalisation is calculated over the features. It is also argues that batch normalisation makes the loss surface smoother. However, we need larger batches, thus increasing the computational costs.

Layer normalisation was introduced as a tool to be able to find the global mean and variance. Here, the mean and variance is the same for all feature dimensional but it is difference for each example in a mini-batch. However, experiments have shown that it does not perform as well with images and better with RNNs.

4 Pytorch CNN

We can see the accuracy and loss curves in the following for the CNN with the described architecture and the default parameters.



Compared to the MLP, we can see that the CNN has a steeper decrease and it converges faster. As expected, a higher accuracy is achieved must faster than with MLP. The final accuracy obtained is 0.79.